Class 7: Clustering and PCA

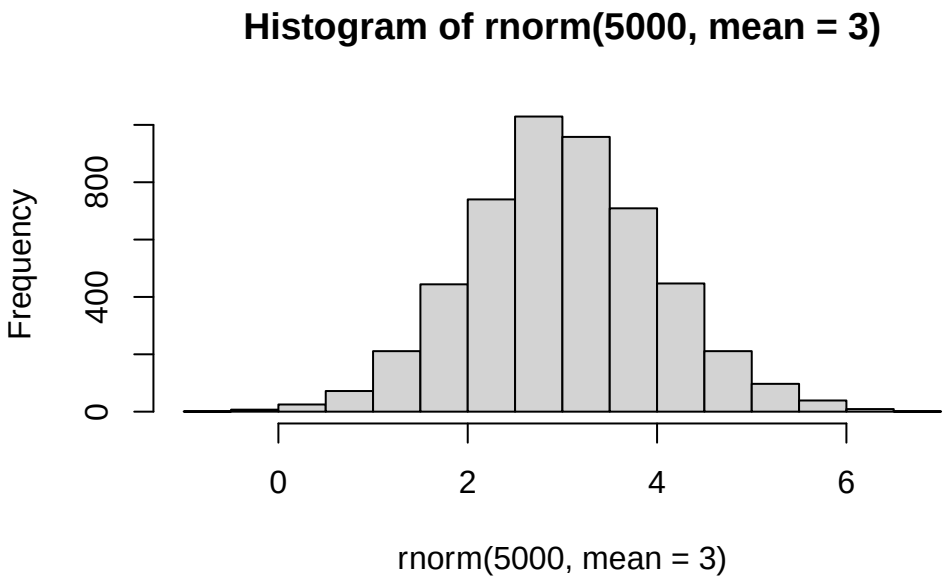
Soomin Park

#Clustering

First, let's amke up some data to cluster so we can get a feel for these methods and how to work with them.

We can use the `rnorm()` function to get random numbers from a normal distribution around a given `mean`.

```
hist( rnorm (5000, mean = 3) )
```



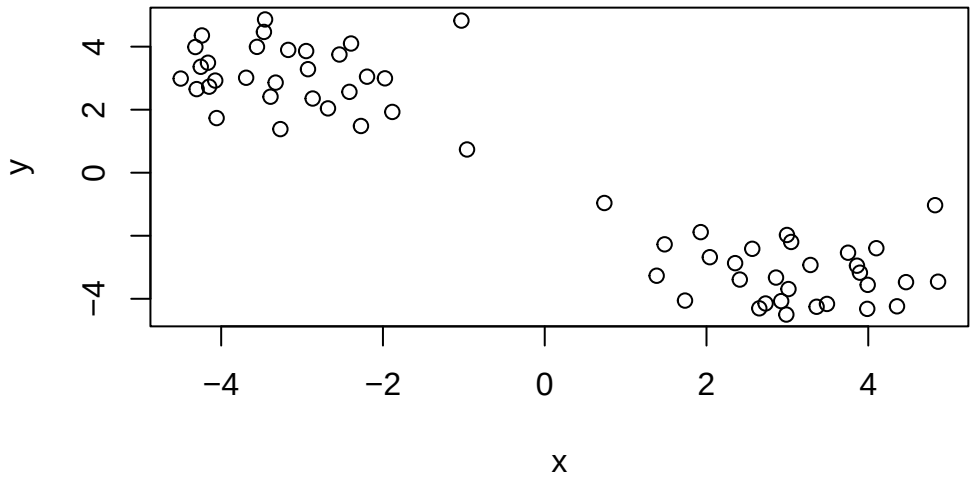
Let's get 30 points with a mean of 3 and another 30 with a mean of -3.

```
tmp <- c(rnorm(30, mean = 3), rnorm(30, mean = -3))
tmp
```

```
 [1]  3.2850702  4.4669143  2.9950625  3.3612283  3.9928609  2.3539495
 [7]  3.9875184  2.6527182  1.7342798  3.8968835  1.3838742  3.8603945
[13]  2.5656900  3.0467789  4.0997421  3.4894377  2.9235103  2.7301491
[19]  0.7371907  3.7500386  2.9870992  1.4823585  4.3554137  1.9284691
[25]  4.8630517  2.8595938  2.4118146  3.0133864  2.0413555  4.8260013
[31] -1.0299560 -2.6796154 -3.6912327 -3.3900302 -3.3267462 -3.4573627
[37] -1.8843436 -4.2384169 -2.2718048 -4.5001137 -2.5379758 -0.9606254
[43] -4.1486712 -4.0735964 -4.1634927 -2.3944035 -2.1966194 -2.4145130
[49] -2.9497470 -3.2681768 -3.1714423 -4.0563433 -4.3030991 -4.3192641
[55] -2.8687371 -3.5574203 -4.2525245 -1.9754156 -3.4714680 -2.9268546
```

Put two of these together:

```
x <- cbind(x = tmp, y = rev(tmp))
plot(x)
```



K-means clustering.

Very popular clustering method that we can use with the `kmeans` function in base R.

```
km <- kmeans(x, center = 2)
km
```

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1 -3.149334  3.069395
2  3.069395 -3.149334

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:
[1] 58.46203 58.46203
 (between_SS / total_SS =  90.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
km$size
```

```
[1] 30 30
```

Use the kmeans() function setting k to 2 and nstart = 0 Inspect / print the results > Q. How mnay points are in each cluster?

```
km$size
```

```
[1] 30 30
```

There are 30 points in each cluster

Q. What 'component' of your result object details

- cluster size?

```
km$size
```

```
[1] 30 30
```

- cluster assignment / membership?

```
km$cluster
```

```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

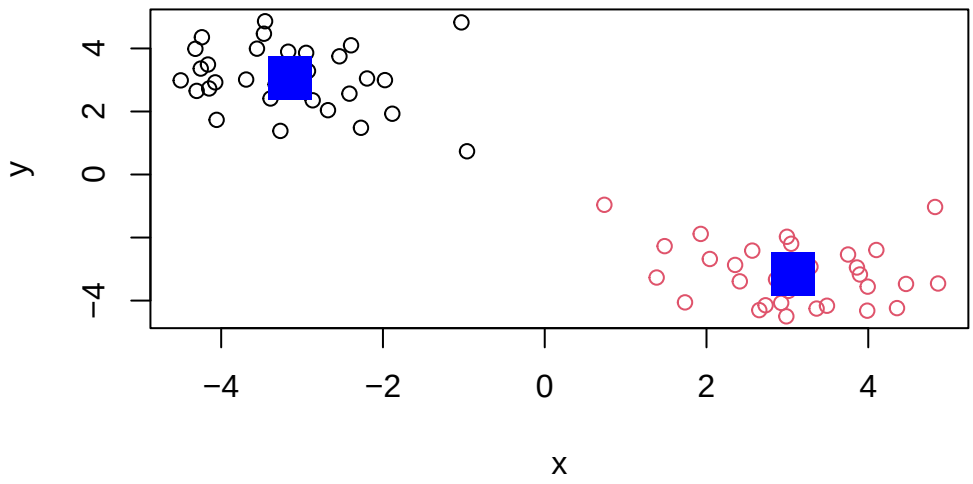
- cluster center?

```
km$centers
```

```
          x         y
1 -3.149334  3.069395
2  3.069395 -3.149334
```

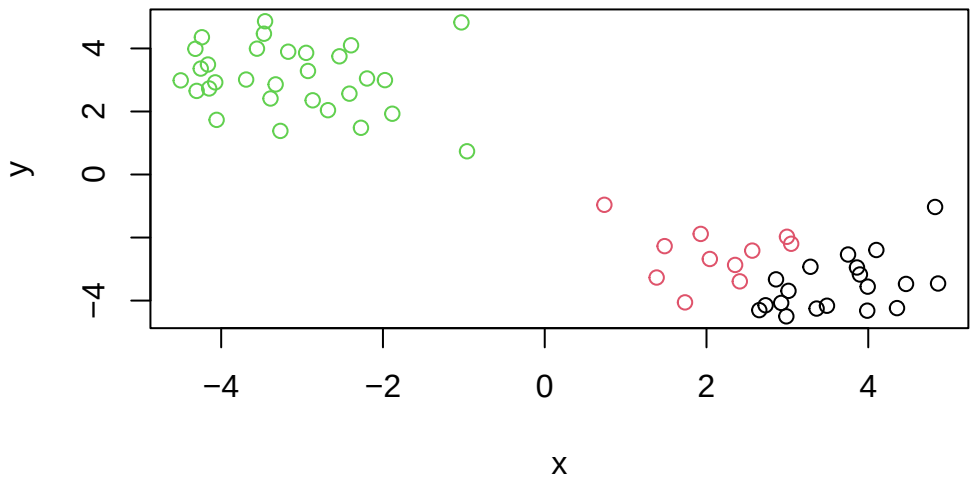
Plot x colored by the kmeans cluster assignent and add cluster centers as blue points.

```
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=3)
```



Q. Let's cluster into 3 groups or same `x` data and make a plot.

```
km <- kmeans(x, center = 3)
plot (x, col=km$cluster)
```



Hierarchical Clustering

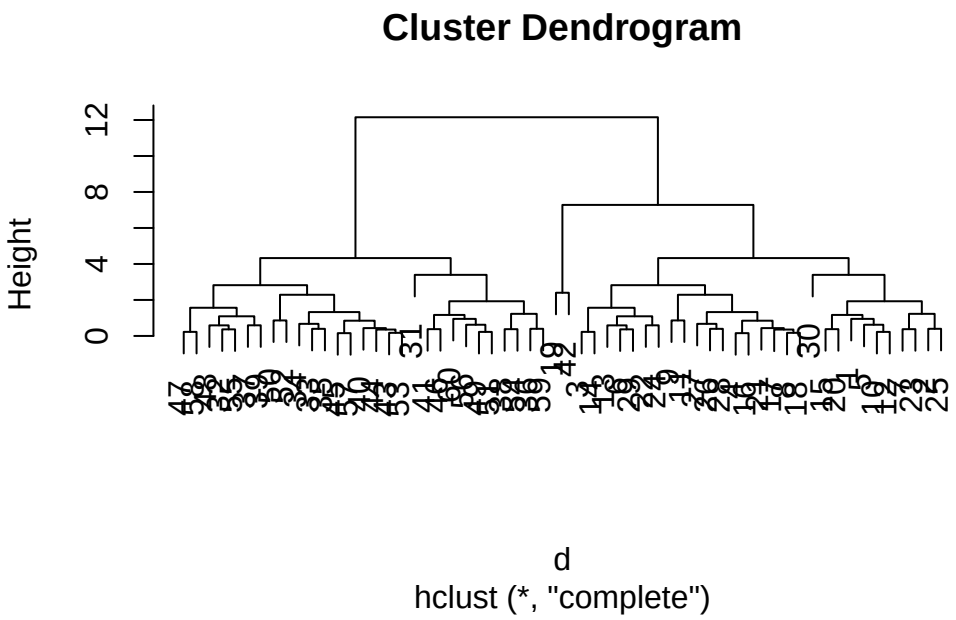
We can use the `hclust()` function for Hierarchical Clustering Unlike `kmeans()` where we could just pass in our data as input, we need to give `hclust()` a "distance matrix".

We will use the `dist()` function to start with.

```
d <- dist(x)
hc <- hclust(d)
hc
```

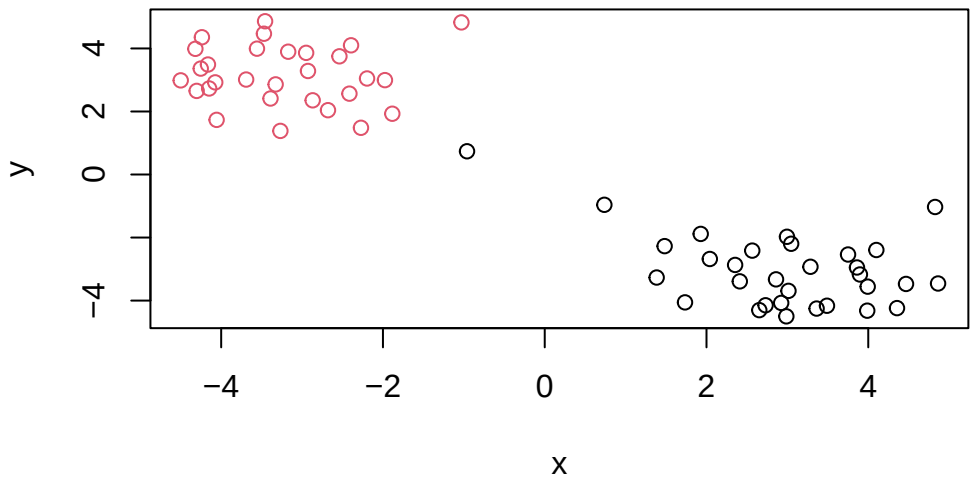
```
Call:
hclust(d = d)

Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

```
plot(hc)
```



I can now "cut" my tree with the `cutree()` to yield a cluster membership vector.

```
grps <- cutree(hc, h = 8)
plot(x, col=grps)
```



You can also tell `cutree()` to cut where it yield "k" groups.

```
cutree(hc, k=2)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Principal Component Analysis (PCA)

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names = 1)
x
```

```
                England Wales Scotland N.Ireland
Cheese              105   103      103        66
Carcass_meat        245   227      242       267
Other_meat          685   803      750       586
Fish                147   160      122        93
Fats_and_oils       193   235      184       209
```

```
Sugars                 156   175      147       139
Fresh_potatoes         720   874      566      1033
Fresh_Veg              253   265      171       143
Other_Veg              488   570      418       355
Processed_potatoes     198   203      220       187
Processed_Veg          360   365      337       334
Fresh_fruit           1102  1137      957       674
Cereals               1472  1582     1462      1494
Beverages               57    73       53        47
Soft_drinks           1374  1256     1572      1506
Alcoholic_drinks       375   475      458       135
Confectionery           54    64       62        41
```

```
# setting row.names = 1 fixes the issues of having 5 columns, first one being x with food
```

Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

```
dim(x)
```

```
[1] 17  4
```

preview the first 6 rows

```
head(x)
```

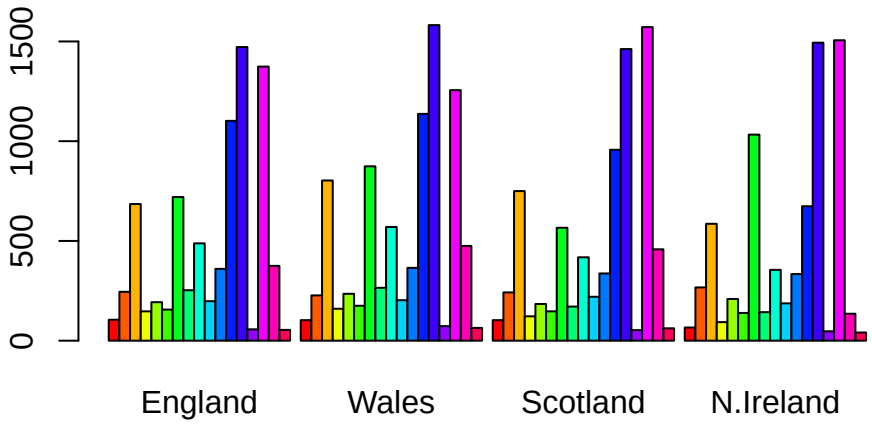
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars            156   175      147       139
```

Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

** I prefer the `x <- read.csv(url, row.names=1)` method as it is shorter and more straightforward. Further, running the first approach code block (i.e. the one with x <- x[,-1]), multiple times may result in errors.**

#Spotting major differences and trends

```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```

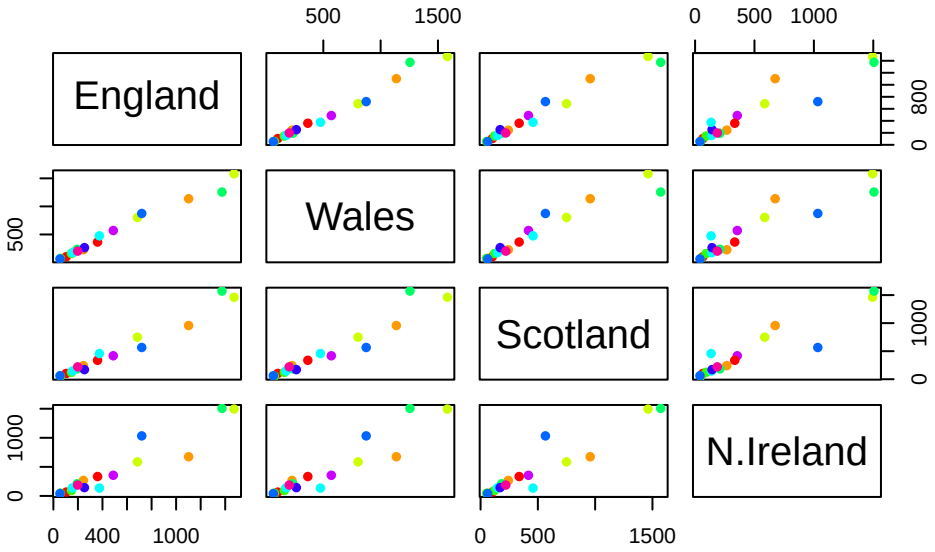


Q3: Changing what optional argument in the above barplot() function results in the following plot?

setting beside=FALSE in the boxplot() code will provide the graph with elements stacked instead of side by side.

Q5: Generating all pairwise plots may help somewhat. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

```
pairs(x, col=rainbow(10), pch=16)
```



The resulting figure shows the relationship between the amount of food consumption in two countries. For example, the bottom plot of the first column explains the food consumption relationship between England and N. Ireland, where the y-axis is N. Ireland and the a-axis is England. Diagonal lines indicate the similarity in food items between two countries being plotted. The points below the diagonal line indicate that that element of food item is consumed by the x-axis country, while the points above the diagonal line indicate that the element of food item is consumed by the y-axis country.

> Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

Blue food item is consumed more often in N. Ireland in comparison to the other countries of the UK, where as orange food item is less frequently consumed in N. Ireland comparatively.

PCA to the rescue

The main PCA function in base R is called `prcomp()` it expects the transpose of our data.

```
# Use the prcomp() PCA function
pca <- prcomp ( t(x) )
summary(pca)
```

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
attributes(pca)
```

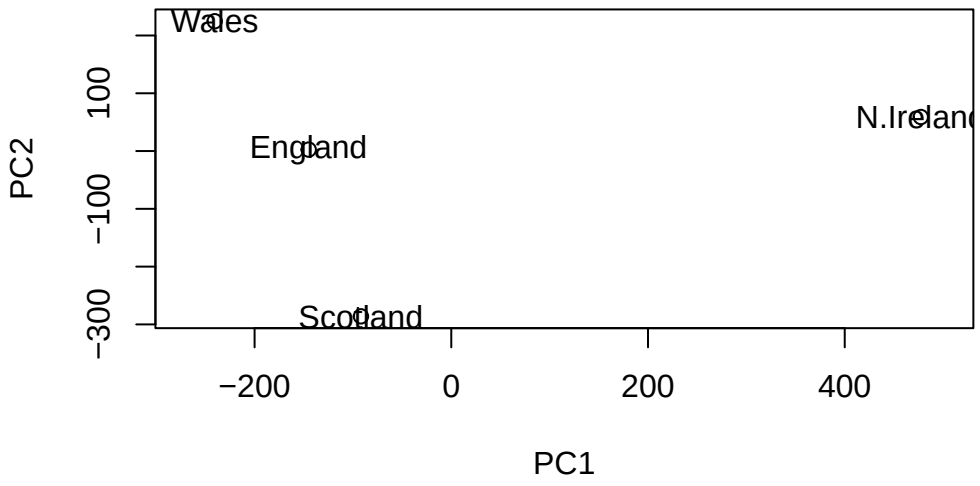
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
pca$x
```

```
                PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

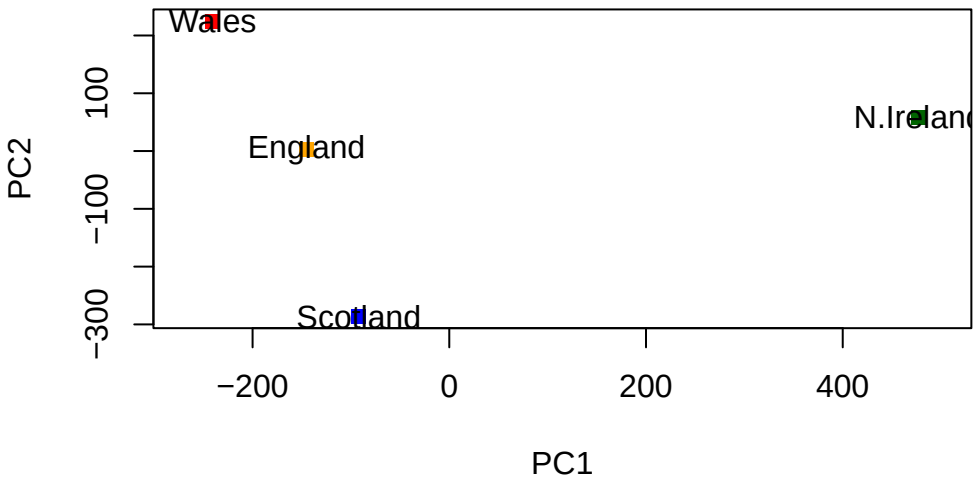
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```



Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

```
plot(pca$x[,1], pca$x[,2], col = c("orange", "red", "blue", "darkgreen"), xlab="PC1", ylab
text(pca$x[,1], pca$x[,2], colnames(x))
```


Wales
England
Scotland
N.Ireland
PC2
100
−100
−300
−200
0
200
400
PC1